EDM and Framework

October 19, 2004

Abstract

In this document, the LPC EDM working group will present our observations and views on the current CMS EDM and framework. These are supported by a small collection of critical use cases, based on our previous experience at collider experiments. These use cases will help to define, illustrate, and capture the desired functionality of an EDM and framework.

Contents

1 Introduction

One of the goals of the LPC (LHC Physics Center) located at Fermilab is to set up an offline environment enabling physicists to engage in physics analysis and to develop software for the CMS experiment. Having such a center in the US will allow the physics community to actively participate in the CMS experiment while minimizing the time the individual physicists has to be present at CERN.

To some extent, this is an experiment by itself. In the past, major physics analysis centers were hosted at the same laboratory that hosted the experiment. In contrast, the LPC will be separated from the experiments by a few thousand miles and several time zones. Fermilab is in a unique position hosting such a center since it currently hosts the two collider experiments CDF and D0 and therefore world experts in various areas of analysis and software are already located at Fermilab.

An important step in this direction is to take currently existing expertise and move it to the CMS experiment. A physicist coming to the LPC needs access to local EDM, framework, and code management experts. He/she should find a well-managed and maintained offline environment which is able to support various analysis and software development projects. The LPC EDM working group was established as a first step to develop such a knowledge base by evaluating the functionality and performance of the current CMS offline framework and event data model (EDM).

In this document, we will present our observations and views on the current EDM and framework. These are supported by a small collection of critical use cases, based on our previous experience at collider experiments. These use cases will help to define, illustrate, and capture the desired functionality of an EDM and framework. A few representative use cases are selected for more detailed analysis, and other use cases that are important but not deemed critical appear in appendix **??**.

In particular, we will evaluate the current CMS system with respect to all infrastructure necessary for a physicist to:

- contribute an algorithm

- develop an algorithm
- perform further event analysis using previous results, taking into account:
  - locating input data
  - configuring of jobs
  - storing results
  - recording parameters (provenance)
- know whether the job completed successfully.
- determine the source of any problems (runtime and logic errors, performance related)
- framework and EDM and how this is used in the trigger system (high level trigger farm)
- framework and EDM and how it supports test beam analysis.
- an EDM that supports multiple objects of the same type and ways to locate them
- an EDM that supports schema evolution.
- problems solved once e.g. one way to track parentage of an event data object
- services provided by the framework
  - shared by all modules
  - how is the interaction of modules handled
  - ntuple and histogram directories are examples
  - database connections are an example
  - access to mass storage is an example
  - various input modules (file, daq...) are examples
  - common User interface is an example

  - means to communicate with other framework or DAQ processes are examples. (robust interprocess communication).
  - centralized facilities that physicist does not need to worry about

- error collection, handling and reporting
- configuration and parameter tracking system
- testing - integration and component
- address issues of required I/O performance

2 Findings

2.1 Reconstruction on Demand

The CMS framework is a specific instance of reconstruction on demand with specific choices made for configuration and rules for what "on the fly" means. These choices do not match the needs of the LPC group. The controls over parameters are not good enough. CMS now has a one-to-one mapping between request for an object and the algorithm that produces it. This is apparently a problem in the input/output specification - it is two simplistic. This is complicated by algorithms that can produce several things and algorithms that produce things that are used in an abstract fashion. Is it possible to create a ROD framework that has adequate controls and configuration and still make it ease to use, or even reasonable?

2.1.1 Reconstruction on Demand - Advantages

- ordering is automatic
- speed - only the things that are needed are run without specific knowledge of paths and flows. While this is often used as an argument to use ROD in the Trigger it depends on the strategy used in the trigger. One possibility is to abort the

trigger once a positive decision is made that an event should be accepted. Another strategy would be to run the entire reconstruction and use the trigger to classify the event for e.g. sending them into different streams. In the later case ROD would not provide performance benefits.

- requires algorithm writers to specify all input data requirements

2.1.2 Reconstruction on Demand - Disadvantages

List of perceived requirements and issues:

1. Resources
   - What is needed? (estimates)
   - Are reservations needed?
   - How is it determined what is needed to run the job?
2. Control
   - Reproducibility
     - not everyone wants the latest and greatest version of code
     - People want a known version of the code
     - controlled update to code
     - the data produced must identify the code that created it
     - No parameters in code
     - parameter sets must be tracked in a repository
   - Transparency
   - Ease of running
   - What was run - perceived difficulty is understanding what is going to be done
   - How is it turned off? Requiring the user to know which items should be turned on or off in a configuration file (.orcarc) is over-burdensome on the user

3. Robustness

    - checks not present, problems encountered at runtime during event processing
    - crashing when algorithm not present when data object is requested

4. Physical Coupling

    - it is harder to minimize the physical coupling between algorithms
    - algorithm code must always be available in order to access already reconstructed event objects
    - The difficulty in locating required shared libraries, or linking the application code is increased

5. Online

6. Dynamic Linking Required (unwise for online?)

7. We would like to see record of what executable program did in a log file after running of the job:

    - want to see (summary) graph of what happened
    - include parameters that triggered paths

8. do not know what triggers the algorithm to go, see explicit controls. (e.g. We saw that calling an iterator triggered the ROD mechanism. All such triggers should be intuitive.)

9. We believe there is a need to be able to check for the existence of event data and it's associated parameters without causing expensive resources to be activated (without our knowledge). There should be a query interface to the EDM

3 Use Cases

The following use cases have been chosen because they have been deemed critical to illustrate the findings above. There are other use cases which are included in appendix **??**.

We have chosen to group the use cases into categories that represent major chronological steps. Requirements for earlier categories also apply to later categories.

- **Trigger** Here we will deal with use cases related to the trigger meaning the use in the filter farm responsible for the high level trigger decisions and for streaming out the data into different data streams. Or related to it the simulation of triggers.
- **Reconstruction** Here we will deal with use cases related to reconstruction. A typical task would be developing a new algorithm and make the persistent results.
- **Analysis** Here we will deal with use cases related to physicists doing an analysis.
- **Data Handling and Data Access** Many functions are probably dealt with by external tools (grid, data catalogs....) but the framework needs means to interact with these tools ... as well as different mass storage systems...

3.1 Trigger

For trigger purposes we need filtering, we need the ability to define a filter path, we need to be able to check the trigger logic which can be fairly complex in a hadron collider environment. We need to be able to monitor the decision, estimate efficiencies and times needed to make a decision (for each module/algorithm). There are strong demands on configuration management since the trigger needs to be in a well defined state. So all configuration information defining this state must be kept. Provenance should be an automatic framework function and the framework needs to provide the means to query and retrieve this information. The use in the trigger puts the most stringent requirements on speed, robustness, and memory usage on the system.

3.1.1 Start algorithm B only if Algorithm A finished successfully

Goal: Start algorithm B only if Algorithm A finished successfully and objects were found
Actor: Framework
Trigger: The code has activated by the presence of a new event.
Description: Want to run Algorithm B only if Algorithm A was executed and finished successfully and produced list of output object with more than 1 entry. (could think of many examples e.g. run vertex algorithm only after tracking ran and tracks were actually found.)

3.1.2 Start algorithm C only if Algorithm A finished successfully and Algorithm B didn't find any object

Goal: Start algorithm C only if Algorithm A finished successfully and Algorithm B didn't find any object
Actor: Framework
Trigger: The code has activated by the presence of a new event.
Description: Want to run Algorithm C only if Algorithm A was executed and finished successfully and produced list of output object with more than 1 entry and Algorithm B didn't find any object that it's searched for (e.g. run MET algorithm (without muon correction) only if Jetfinder ran successfully but no muon was found in the event.)

3.1.3 Use RecCollection created by an algorithm correctly in a second algorithm.

Goal: Use RecCollection created by an algorithm correctly in a second algorithm.
Precondition: Algorithm 1 has created a RecCollection A. Due to a detector failure, the RecCollection is empty or incomplete.
Actor: Algorithm 2
Description: Algorithm 2 should be able to query the RecCollection A to see if there were errors in its construction.
Example: Unpacking of a quadrant of a calorimeter fails due to

data corruption. The resulting collection of towers is incomplete. The Missing ET trigger algorithm needs to be able to treat this case differently than an empty or complete list of towers.

3.1.4 On line Monitoring of data quality

Actor: Monitoring Framework writer
Goal: on line monitoring of data quality. This adds quite a few requirements to the framework. Here the Processes needs to be able to receive events in real time (framework function/input module/algorithm). Monitor must be able to select events passing a specific trigger or pass a given filter pass. Analyzing process needs to communicate with other processes (robust interprocess communication) .

Description:

1. e.g. collector processes (collecting histograms from many analyzers.)
2. e.g. error logger error receiver
3. e.g. run control (might be the error receiver independent from the framework doing this communication)

3.2 Reconstruction

3.2.1 Creating hierarchical objects

Goal: Build a high-level (HL) object from a set of lower-level (LL) objects of varying types.
Precondition: the actor has access to the lower-level objects by some means beyond the scope of this use case.
Actor: Algorithm writer
Description: A HL object can be composed of multiple LL objects. An algorithm writer will have to build up the HL object in such a way that a downstream physicist can access the LL objects as described in the previous use case. There should be standard EDM tools that allow the creation of linkages from the HL object to the LL

objects. The actor should not have to know the LL object ID's, but should be able to just "add" the LL object to the HL object. In particular, if the lower object does not have independent, standalone status in the event record, but is an element of some other object (e.g. a hit within a collection of hits), the user shouldn't even have to know the element number. If the LL object is itself composed of multiple subordinate objects (as a segment on a muon track is composed of individual hits), then its linkages should be naturally transferred to the HL object.

3.2.2 Creating a new algorithm (Vertex fitter)

Goal: Creates a new algorithm e.g. a Vertex fitter and wants to store the new object as well as updated tracks in the data stream. If there is already such an object he wants to add his (new version).
Actor: Algorithm writer
Description:

1. needs to get a list of tracks, access to track parameters and covariance matrix.

2. needs to be able to select tracks that pass a given set of cuts.

3. needs to associate the tracks used in the fit with the vertex object, How is association handled in general?

4. needs to store new tracks /track collection since the vertex constraint improves parameter and error estimates.

3.2.3 Create new algorithm and make persistent

Goal: Create new persistent Algorithm
Actor: Physicist
Trigger: The code was activated by the presence of a new event.
Description: Write a reconstruction algorithm to calculate the total energy deposited in the calorimeter for an event. Make the energy persistent and write it into the DST.

3.3 Analysis

3.3.1 Find out which algorithms have been run already within an event

Goal: From within the running framework program processing a given event, find out which algorithms have already been run with which version and which configuration within the event.
Actor: Physicist
Trigger: My code was activated by the presence of the a new event.
Description: The event will contain the results of processing by various algorithms. Within an event, it is needed to query which algorithms have been run already both to avoid rerunning particular algorithms and to determine if dependent algorithms can be started. To make the determination, the parameters used to configure particular algorithms are needed.

3.3.2 Print to Standard Output the Pt of tracks

Goal: Print to Standard Output the Pt of tracks
Actor: Physicist
Trigger: My code has activated by the presence of a new event.
Description: In the current event, I will ask for all those tracks found by algorithm named X version n, with completely unique and unambiguous configuration Y. There must be either 0 or 1 collections of tracks associated with this run of the algorithm. If this version of this algorithm has not previously been run with this configuration, I will print a message saying it has not been run. If this algorithm has been run, I will iterate through all those tracks, printing to standard output the transverse momentum of the track.

3.3.3 Locating EMobjects and references Tracks

Goal: As part of the E/p calibration calculation for a calorimeter, locate the EMobjects and find the tracks associated with them.
Actor: Physicist
Trigger: The code has activated by the presence of a new event.
Description: Using the event that I was given, I will ask the event for a collection of EMobjects created by clustering algorithm X out

of the first pass of reconstruction. For each EMobject in this collection, I will get the list of references to tracks associated with this EMobject. If the reference list is empty, then this EMobject should be skipped. For each references, I want to go back to the event and resolve it to the actual track instance. If the track instance is not in the event, then this reference will be skipped. I will use the track instance to get the track momentum and cluster energy and do the calculation.

3.3.4 Comparing two track collections

Goal: Do a tracking performance study, comparing efficiency of two algorithm.
Precondition: A particle gun was used to generate the input sample. Each event in this sample was generated requesting 10 tracks.
Actor: Physicist
Trigger:
Description: Create an event file, reconstructing data with algorithm X, version 1. Read the event file into another process that reconstructs the data again using algorithm version 2. This organization matches the process involving keeping the level-3 trigger tracks and then having production (reconstruction) add tracks also.

The analysis program, reads in the events, gains access to the collection of reconstructed Tracks from algorithm X, versions 1 and 2 from each event. Compare the number of tracks in each of the two different collections.

Note: What is required here? Generating two differently named classes? Using an abstract interface for tracking (fixed API) with creation helped by object factories? Is there a concept of transient representations of tracks?

3.3.5 Navigating hierarchical objects

Goal: From a high-level object, learn about the lower-level objects that compose it.
Precondition: the actor has been handed the high-level object by some means beyond the scope of this use case.

Actor: Physicist
Trigger:
Description: A high-level (HL) object can be composed of multiple lower-level (LL) objects of different types (which are thus represented by different classes). A specific example of a HL object is a muon, which may be composed of a) the track found in the muon system, b) the segments that form those tracks, c) the hits that form the segments, d) silicon hits associated with the track, or possibly an entire silicon track found by other means and associated with the muon, e) calorimeter towers that the muon has passed through. The physicist should be able to work downward through the hierarchy to access the LL objects, i.e. there should be easy access to the segments, and in turn from the segments to the hits that form them, for instance. To achieve this, there should be a straightforward set of accessors that return the LL objects. Access to the LL objects from the HL object should not require any knowledge of object ID's of the former.

3.3.6 Run the detector simulation and reconstruction with a different geometry

Goal: Wants to run the detector simulation and reconstruction with a different geometry e.g. to simulate a test beam set up. (e.g. misaligned better aligned, simpler, new more realistic) how is this configured and how is this communicated to the different applications. (geometry stored in the data stream?)
Actor: Developer
Description: How is configuration managed (e.g. standard configuration stored somewhere so that we only need to worry about differences from the standard.)

3.3.7 Create more Monte Carlo Data

Goal: have a given (MC) data set and wants to create more statistics.
Actor: Physicist
Description:

1. Is all the information available to recreate/increase the data set. (data cards, random seeds, software versions.....)

2. How does the physicist get to this information? (query the event , external data base)

3.3.8 Analysis without framework

Goal: use familiar tool to look at data
Actor: Physicist
Description:
wants to use a tool that he is familiar with to look at the data. Is it possible to browse/look at the data without depending on the full framework/EDM? e.g. one approach would be the case where you just load in some shared libraries in root which then would allow you to look at the data with the provided TBrowser.

This issue is also described in document [2] CD-doc-435.

3.3.9 User Interface: Interacting with the framework

Goal: needs unique interface as part of the framework.
Actor: Physicist
Description: wants to interact with the framework/reconstruction executable e.g. get a list of available commands and their syntax as well as the default parameters. He wants to find the same environment independent of the specific program he is is dealing with.

1. needs common user interface as part of the framework.

2. needs help and show functions for the commands.

3. needs reasonable defaults for all necessary module/algorithms that have to be initialized. (what happens if you specify no input parameters at all?).

4. list available analysis module/algorithms

5. list current and default settings

6. provide Documentation what this parameters actually mean (help and show commands)

7. provide check of valid ranges for parameters
8. configure path, decision tree
9. needs to store the commands he typed in (last settings, history) to use in a batch job afterward and to document his work.
10. informational message when there is an incorrect input parameter or incorrect combinations thereof. Ignoring the line or a seg fault at run time are not acceptable.
11. For each event optionally store decision tree and make it available for printing.

Comments:

- putting parameters in hidden files (.orcarc) can be very confusing to new users.
- One way to interact with the system is an interactive shell (e.g. CDF framework, ROOT) which gives the user the possibility to learn what commands and module/algorithms are available. Some users like the try and error approach to find out what works.
- There seems to be a lot of traffic on the mailing lists regarding the right set of parameters.
- At the moment it seems to be necessary to look into the source code to figure out what some of the parameters mean.

3.3.10 Interaction with physics object/Algorithm

Goal: Let the object describe itself using a well defined interface
Actor: Physicist
Trigger:
Description: wants to interact with a physics object. He wants the object to provide him with a (print) method to list all data members, all available accessors, all input parameters and all available functions.

3.4 Data Handling and Data Access?

3.4.1 Availability of datasets

Goal: Want to know what data sets are available:
Actor: Physicist
Trigger: wants to start an analysis and wants to know what is already available.
Description:

1. How does he find out?

2. Wants to know what is in the data sets: Browse the data set, dump the objects, display the events, find out how the data set was created, version of code etc., database entry being used or run conditions ....

3.4.2 Create self-contained dataset

Goal: create a self contained data set
Actor: Physicist
Trigger: Long trip in airplane without network
Description: wants to download a dataset to his laptop or dvd and analyze while on the run. Is it possible to create a stand alone self contained data set which is meaningful without access to e.g. data bases. What is the price user has to pay? (e.g. no immediate access to updated database information, no way to reprocess?) how can you check that you got everything (all files etc.) besides running the job and waiting until it crashes.
Comment Another similar example could be a specific grid job where the process might be required to execute in a sandbox with no access to data base transactions while running.So the job has to import its entire environment and input data at start up time.

3.4.3 Combining data from different processing streams

Goal: Runs his job using many processes in parallel on a farm needs to combine the data produced by the different processing streams.

Actor: Physicist
Trigger: make data set available for collaborators
Description: Runs his job using many processes in parallel on a farm needs to combine the data produced by the different processing streams.
Comments:

Hans: Finally gave up on this all together. Since I decided it's not worth the trouble. Need to write up what I did and the troubles encountered. Could fill pages with email exchanges and the things I tried.

- unless running a database Pool doesn't provide concurrency so with a plain file catalog you can't write into the same federation in parallel.Not an expert in setting up a database. That means after each processing step one needs to attach your files etc. to the master Poolfile catalog.

- It adds many more steps to the data handling process (more than doubles need to add a diagram). Makes processing on a farm much more complicated and leads to inefficiencies running the batch system.

- Behavior of e.g. the init data set step is different in different (OSCAR) versions of the code. Some version we didn't get t it working at all, in some versions you had to initialize a generator (what does that have to do with initializing a data set??) and you actually needed to put all this unnecessary stuff in the orcarc file. When not processing an orcarc file at least the program should respond differently from just crashing. There was a version that actually was able to initialize the data set without having to initiate other stuff but then the OSCAR version after that couldn't do that anymore.

- The next step writing independent streams never worked (for us) it was recognized as a bug in cobra and reported to the developers.

- officially gave up.

3.4.4 Make highly condensed datasets available to collaborators

Goal: Create a highly condensed dataset and make it available to collaborators
Actor: Physicist
Trigger: prepare for an analysis
Description: For an analysis a highly condensed data set is needed that can be spun through fairly quickly. The analysis group has agreed on the selection criteria and cuts that go into this sample.

1. Needs to select events and write into a new output stream (event selection, filtering, setting a filter path)
2. Needs to drop banks/objects which are of no interest for the analysis
3. Makes the data set available.
4. Related to configuration and configuration management/data base access. Provenance: The data set should include all the information about how it was created.

4 Detailed analysis of selected use cases

CMS software as well as the software libraries that CMS code depends on are still changing rapidly. Therefore we will give the specific versions we used. Some of the comments might not apply to later versions of the software anymore as problems get addressed by the developers.

4.0.5 Analysis of use case 3.3.2: Print to Standard Output the Pt of tracks

At the time of the execution of this use case, only one version and one configuration for each algorithm was supported. The specific ORCA version that was used is $ORCA\_8\_0\_1$. Therefore, only the default configuration and version could be used, which simplified the use case.

I was unable to skip an event that did not use the chosen algorithm. This is because, at the time of this use case, "reconstruction on demand" forces the algorithm to be run if it had not previously been run. It is my understanding that in a subsequent release of COBRA/ORCA, that there will be a mechanism to disable reconstruction on demand.

The most difficult part of the use case was determining the set of shared libraries that needed to be loaded with the executable. Fortunately, I could use the existing SCRAM BuildFile for the readDST executable. However, this caused many more shared libraries to be linked than were needed. Determining the minimal set of libraries needed was not done.

The code for this use case was, except for the output statements, almost a subset of readDST, which made it easier than it would otherwise have been.

The main code for this use case is in appendix B.

4.0.6 Analysis of use case 3.2.3: Create a new algorithm to find the total energy deposited in the calorimeters for an event. Make the total energy persistent and write it into the DST.

The specific ORCA version that was used is $ORCA\_8\_0\_1$.

The algorithm uses the existing class CaloRecHit for reconstructed calorimeter hits. However, CaloRecHits does not inherit from RecObj, i.e., it is not a "standard" reconstructed object. Thus, it cannot be used as a component of the new algorithm in the standard manner.

The most difficult part of the use case was determining the set of shared libraries that needed to be loaded with the executable. Fortunately, I could use the existing SCRAM BuildFile for the writeDST executable. However, this caused many more shared libraries to be linked than were needed. Determining the minimal set of libraries needed was not done.

The code for this use case was modeled after the code for writeDST, which made it easier than it would otherwise have been.

The main code for this use case is in appendix C.

A Glossary

A.1 General

EDM Event Data Model: An Event Data Model(EDM) is a set of software components which provide a mechanism for managing data related to an event (i.e. a collision) within a program. An EDM is not merely a persistency mechanism, nor is it an input/output mechanism or a file format although it is related to these things.

Use Case A way to capture and model known functional requirements. This is done in a story-like format which makes it easy to understand and to relate to the users own experience or interaction with the system for which the requirements are collected. Use cases are only part of the requirements they don't detail e.g. interfaces, speed requirements, type safety.... In fact some requirements are very difficult to express in form of use cases.

Actor A person or computer program/system that interacts with our system for the purpose of completing some goal.

Goal A task or job that an actor wants to complete or accomplish.

Stakeholder The entity that benefits or suffers as a result of actors making use of the system.

Narrative Short paragraphs describing the things that an actor does to accomplish one goal. A set of these is a collection of activities that represents what the system is going to do for us. They will be reviewed and turned into a set of more formal use cases.

Framework provides a mechanism for constructing programs (typically: triggering, filtering, reconstruction, analysis, and event display) programs from independent modules. Often, an EDM has no knowledge of (or dependence upon) a framework,and a framework has little or no knowledge of the EDM.

Algorithm/module

Observer pattern (active passive

A.2 Acronyms of CMS Object-Oriented Projects

Table 1 presents acronyms and their translations for several of the major CMS Object-Oriented projects. Detailed descriptions of these projects can be found at the CMS Object-Oriented home page [1].

Table 1: Selected acronyms for CMS Object-Oriented Projects

Acronym	Meaning
ORCA	**O**bject-oriented **R**econstruction for **C**MS **A**nalysis
IGUANA	**I**nteractive **G**raphical **U**ser **ANA**lysis
IGUANACMS	**I**nteractive **G**raphical **U**ser **ANA**lysis for **CMS**
DDD	**D**etector **D**escription **D**atabase
OSCAR	**O**bject-Oriented **S**imulation for **C**MS **A**nalysis and **R**econstruction
FAMOS	CMS Fast Simulation
COBRA	**C**oherent **O**bject-Oriented **B**ase **R**econstruction and **A**nalysis
SCRAM	**S**oftware **C**onfiguration, **R**elease **A**nd **M**anagement

B Source code for use case 3.3.2

B.1 BuildFile

```
# This is much more than is necessary for th BuildFile
# No attempt was made to use only a minimal set of libraries
 <external ref=pi use="AnalysisServices/AIDA_Proxy">
 <external ref=Aida>

<environment>
  <lib name=EcalPlusHcalTower></lib>
  <lib name=CaloCluster></lib>
  <lib name=HcalRealistic>
```

```
 <Group name=CaloHitReader>
 <Group name=CaloRHitWriter>
 <Group name=CaloRHitReader>
 <Group name=TriggerPrimitiveWriter>
 <Use name=Calorimetry>

  <lib name=MuonUtilities>
  <lib name=CommonNavigation>
  <lib name=MuonReconstruction>
  <group name=MuonTrackFinder>
  <group name=L1TRIGGER>
  <lib name=PixelReconstruction>

 <lib name=MuonUtilities>
 <lib name=MuonIsolation>

<use name=Trigger>
<use name=TrackerReco>

 <external ref=gsl>
 <Group name=MuonInternalReco>
 <Use name=Muon>

 <lib name=PrincipalVertexReco></lib>
 <lib name=CARFVertex></lib>
 <group name=LeastSquaresVertexFit>
 <use name=Vertex>

 <lib name=PixelTrackFinder>
 <group name=TkTrackReader>
 <use name=TrackerReco>

 <lib name=AdvancedJets></lib>
 <lib name=PersistentJet></lib>
 <lib name=KtJets></lib>
 <group name=JetsFromGeneratorParticles>
 <group name=JetsFromEcalPlusHcalTowers>
 <use name=Jets>

  <lib name=L1GlobalTrigger>
  <lib name=L1Trigger>
  <lib name=L1PersistentTrigger>

 <Group name=EgammaBase>
 <Group name=EgammaSetup>
```

```
  <Group name=EgammaEscale>
  <lib name=ClusterTools>
  <lib name=EgammaBasicClusters>
  <lib name=EgammaSuperClusters>
  <lib name=EgammaPreshower>
  <use name=ElectronPhoton>

  <Group name=GeneratorCARFReader>
  <External ref=COBRA Use=GeneratorInterface>

  <group name=RecReader>
  <External ref=COBRA Use=CARF></Use>

  <bin file=UseCase.cpp>To print the DST contents</bin>
</environment>
```

B.2 UseCase1.cpp

```
#include "Utilities/Configuration/interface/Architecture.h"
#include "Utilities/Notification/interface/PackageInitializer.h"
#include "Examples/ExUtils/interface/ExDumpRunEventRecReader.h"

#include "Utilities/Notification/interface/Observer.h"
#include "CARF/Reco/interface/RecCollection.h"
#include "CARF/Reco/interface/RecQuery.h"
#include "Utilities/Notification/interface/PackageInitializer.h"
#include "Utilities/GenUtil/interface/CMSexception.h"
#include "Utilities/Notification/interface/TimingReport.h"

#include "Calorimetry/EcalPlusHcalTower/interface/EcalPlusHcalTower.h"
#include "CommonReco/PatternTools/interface/TTrack.h"
#include <iostream>
#include <iomanip>

class G3EventProxy;

class UseCase : private Observer<G3EventProxy*>
{

public:

  UseCase() : theTkCollection(0) {
    Observer<G3EventProxy*>::init();
  }
  ~UseCase() {
```

```
    delete theTkCollection;
  }
private:
  virtual void analysis();

  void upDate(G3EventProxy* ev) {
    if (ev != 0) {
      analysis();
    }
  }

  RecCollection<TTrack>* theTkCollection;
};

void UseCase::analysis() {
  if (theTkCollection == 0) {
    theTkCollection =
    new RecCollection<TTrack>( RecQuery("CombinatorialTrackFinder","0.0"));
  }
  cout<<"Found "<<theTkCollection->size()<<" Tracker tracks."<<endl;
  for(RecCollection<TTrack>::const_iterator it=theTkCollection->begin();
    it!=theTkCollection->end(); it++) {
    GlobalVector p = (*it)->momentumAtVertex();
    cout << setprecision(4) << showpoint << "p="
         << setw(8) << p.mag() << " GeV, pt="
         << setw(8) << p.perp() << " GeV, theta="
 << setw(8) << p.theta() << " rad, phi="
 << setw(8) << p.phi() << " rad, eta="
 << setw(8) << p.eta() << endl;
  }
}

PKBuilder<UseCase> eventAnalyser("UseCase");
```

B.3 orcarc

```
CARFVerbosity = silent
CobraSignalHandler = false
FinalAbort = true
ForceExit = true
TextColor = false
TimingReport = true

FirstEvent = 0
```

```
MaxEvents = 2

#Input File catalog is defaulted
InputCollections = /System/myDST/h300eemm

CaloDataFrame:SuppressionStyle = 2
TFileAdaptor = true
NoAncestor = true
```

C Source code for use case 3.2.3

C.1 BuildFile

```
# This is much more than is minimally needed.
# No attempt was made to get a minimal set.
<environment>
  <lib name=EcalPlusHcalTower></lib>
  <lib name=CaloCluster></lib>
  <lib name=CaloEnergy></lib>
  <Group name=CaloRecHitReader>
  <Use name=Calorimetry>

  <Group name=MuonInternalReco>
  <Use name=Muon>

  <lib name=PrincipalVertexReco></lib>
  <lib name=CARFVertex></lib>
  <group name=LeastSquaresVertexFit>
  <use name=Vertex>

  <lib name=PixelTrackFinder>
  <group name=TkTracks>
  <use name=TrackerReco>

  <lib name=AdvancedJets></lib>
  <lib name=PersistentJet></lib>
  <lib name=KtJets></lib>
  <group name=JetsFromGeneratorParticles>
  <group name=JetsFromEcalPlusHcalTowers>
  <use name=Jets>

  <Group name=L1GLOBAL>
  <Use name=Trigger>
```

```
  <Group name=EgammaBase>
  <Group name=EgammaSetup>
  <Group name=EgammaTracks>
  <Group name=EgammaEscale>
  <Group name=EgammaSelections>
  <Group name=EgammaL1>
  <lib name=ClusterTools>
  <lib name=EgammaBasicClusters>
  <lib name=EgammaSuperClusters>
  <lib name=EgammaPreshower>
  <lib name=EgammaOfflineReco>
  <lib name=EgammaCalibObject>
  <Group name=EgammaElectron>
  <Group name=EgammaMC>
  <Group name=EgammaPhoton>
  <use name=ElectronPhoton>

  <Group name=GeneratorCARFReader>
  <External ref=COBRA Use=GeneratorInterface>

  <Group name=HLTXML>
  <Group name=HLT_EGamma>
  <Group name=HLT_Muon>
  <Group name=HLT_JetMet>
  <lib name=HLTbtau>
  <lib name=TkPartialReco>
  <use name=HLT>

  <group name=BTagCombined>
  <use name=BReco>
  <lib name=D0PhiSecondaryVertexReco>

  <group name=METreconstruction>
  <use name=MET>


  <group name=RecReader>
  <External ref=COBRA Use=CARF></Use>

  <bin file=UseCase2.cpp>To write total calorimeter energy</bin>
</environment>
```

C.2 classes_def.xml

```
<lcgdict>
```

```
  <class name="CaloEnergy"/>
</lcgdict>
```

C.3 classes.h

```
#include "Utilities/Configuration/interface/Architecture.h"
#include "Calorimetry/CaloEnergy/interface/CaloEnergy.h"

namespace {
  namespace {
  }
}
```

C.4 CaloEnergy.h

```
#ifndef CaloEnergy_h
#define CaloEnergy_h

#include "CARF/Reco/interface/RecObj.h"

class CaloEnergy : public RecObj
{
public:

  CaloEnergy() : energy_(0) {}
  explicit CaloEnergy(double e) : energy_(e) {}

  virtual ~CaloEnergy(){}

  double energy() const {return energy_;}

private:
  double energy_;
};

#endif
```

C.5 CaloEnergyBuilder.cpp

```
#include "Utilities/Configuration/interface/Architecture.h"
#include "Calorimetry/CaloRecHit/interface/CaloRecHit.h"
#include "Calorimetry/CaloEnergy/src/CaloEnergyBuilder.h"
```

```
#include "Calorimetry/CaloEnergy/interface/CaloEnergy.h"
#include "Calorimetry/CaloCommon/interface/CaloItr.h"

#include "CARF/Reco/interface/ParameterSetBuilder.h"
#include "CARF/Reco/interface/ComponentSetBuilder.h"
#include "CARF/Reco/interface/RecConfig.h"
//--------------------------------------------------------------------

CaloEnergyBuilder::CaloEnergyBuilder(const RecConfig& config) :
  RecAlgorithm<CaloEnergy>(config) {
  setMeAsDefault();

  // initialize the reconstruction parameters
  HCALcutoffEnergy= parameter<double>("HCALcutoffEnergy");
  ECALcutoffEnergy=parameter<double>("ECALcutoffEnergy");

  std::cout << std::endl <<
  "CaloEnergyBuilder:HCALcutoffEnergy = " << HCALcutoffEnergy << std::endl
       << "CaloEnergyBuilder:ECALcutoffEnergy = " << ECALcutoffEnergy
       << std::endl  ;
}
//--------------------------------------------------------------------

CaloEnergyBuilder::~CaloEnergyBuilder() {
}

//--------------------------------------------------------------------
RecConfig
CaloEnergyBuilder::defaultConfig() {

  Name name("CaloEnergyBuilder");
  Version version("1.0");

  ParameterSetBuilder param_set_builder;
  const double default_tolerance = 0.001; // tolerance for parameter comparison
  param_set_builder.addParameter("HCALcutoffEnergy",0.0,default_tolerance);
  param_set_builder.addParameter("ECALcutoffEnergy",0.0,default_tolerance);

  ComponentSetBuilder component_set_builder;
  return RecConfig(name,version,param_set_builder.result(),
                        component_set_builder.result());
}

//--------------------------------------------------------------------
```

```
void CaloEnergyBuilder::reconstruct() {
  double energy=0.0;

  CaloItr<CaloRecHit> hcalhit(SuId("HR","HCAL","01"));
  while(hcalhit.next()) {
    if (hcalhit->energy() > HCALcutoffEnergy) energy += hcalhit->energy();
  }
  CaloItr<CaloRecHit> ebryhit(SuId("CR","EBRY","01"));
  while(ebryhit.next()) {
    if (ebryhit->energy() > ECALcutoffEnergy) energy += ebryhit->energy();
  }
  CaloItr<CaloRecHit> efryhit(SuId("CR","EFRY","01"));
  while(efryhit.next()) {
    if (efryhit->energy() > ECALcutoffEnergy) energy += efryhit->energy();
  }
  addObjToReconstructor(new CaloEnergy(energy));
}
//------------------------------------------------------------------------
#include "Utilities/Notification/interface/PackageInitializer.h"
#include "CARF/Reco/interface/RecBuilder.h"
static PKBuilder<RecBuilder<CaloEnergyBuilder> >
       calo_energy_builder_factory("CaloEnergyBuilderBuilder");
```

C.6 CaloEnergyBuilder.h

```
#ifndef CaloEnergyBuilder_h
#define CaloEnergyBuilder_h

#include "CARF/Reco/interface/RecAlgorithm.h"
class CaloEnergy;

class CaloEnergyBuilder : public RecAlgorithm<CaloEnergy>
{
public:

  static RecConfig defaultConfig();

  explicit CaloEnergyBuilder(const RecConfig& config = defaultConfig());

  virtual ~CaloEnergyBuilder();

  void reconstruct();
```

```
private:

  double HCALcutoffEnergy;
  double ECALcutoffEnergy;
};

#endif
```

C.7 UseCase2.cpp

```
// include a helper class to print run/event numbers
#include "Utilities/Configuration/interface/Architecture.h"
#include "Utilities/Notification/interface/PackageInitializer.h"
#include "Examples/ExUtils/interface/ExDumpRunEventRecReader.h"

#include "Utilities/Notification/interface/Observer.h"
#include "CARF/Reco/interface/RecCollection.h"
#include "CARF/Reco/interface/RecQuery.h"
#include "Utilities/Notification/interface/PackageInitializer.h"
#include "Utilities/GenUtil/interface/CMSexception.h"

#include "Calorimetry/CaloEnergy/interface/CaloEnergy.h"

#include <iostream>

class G3EventProxy;

class UseCase : private Observer<G3EventProxy*>
{

public:

  UseCase() : theCaloCollection(0) {
    Observer<G3EventProxy*>::init();
  }

  ~UseCase() {
    delete theCaloCollection;
  }

  virtual void analysis();

private:
```

```
  void upDate(G3EventProxy* ev) { if (ev != 0) analysis(); }

  RecCollection<CaloEnergy>*      theCaloCollection;
};

void UseCase::analysis() {

  if (theCaloCollection == 0) {
     theCaloCollection =
     new RecCollection<CaloEnergy>( RecQuery("CaloEnergyBuilder","1.0"));
  }
  cout << "Found " << theCaloCollection->size() << " CaloEnergy." << endl;

  double energy = 0.0;
  for(RecCollection<CaloEnergy>::const_iterator it=theCaloCollection->begin();
    it!=theCaloCollection->end(); ++it) {
     energy += (*it)->energy();
  }
  cout <<
   "Total energy" << setw(8) <<
    setprecision(4) << energy << " GeV" << endl;

}

PKBuilder<UseCase> myEventAnalyser("UseCase");
```

C.8 orcarc

```
InputCollections =/System/Digis/h300eemm
MaxEvents = -1

OutputDataSet = /System/myDST/h300eemm
OutputRunNumber = 5

CMSRandom:Seeds = 40 3

#----
GoPersistent = true

DBPopulator:CommitInterval = 10
```

```
#---- the stuff for the Digis to add ----
Events:Location = Events
Collections:Location = Events

#--- the name of the RecAlgorithm to store the output from
CaloEnergyBuilder:Persistent = true
##--- where to store it ------------------------
CaloEnergyBuilder:Location = DST
```

References

[1] http://cmsdoc.cern.ch/cmsoo/cmsoo.html

[2] http://computing.fnal.gov/cgi-bin/docdb/public/DocumentDatabase/ CD-doc-435-v1